DOCUMENT RESUME

ED 087 425 IR 000 169

AUTHOR Braun, Peter H.
TITLE Reflections on CAI Language Design.
PUB DATE Apr 73
NOTE 14p.; Paper presented at the Association for Educational Data Systems Annual Convention (New Orleans, Louisiana, April 16 through 19, 1973)

EDRS PRICE MF-$0.65 HC-$3.29
DESCRIPTORS *Comparative Analysis; *Computer Assisted Instruction; Computer Programs; Computer Science; Programing; *Programing Languages
IDENTIFIERS AEDS; APL; Association for Educational Data Systems; BASIC; *Course Authoring Language; COURSEWRITER; Meta Language

ABSTRACT

The need for a high level computer-assisted instructional (CAI) course authoring language is discussed and the capabilities of COURSEWRITER, BASIC, and APL for meeting this need are compared. The demand for such a language is first documented and some historical trends of CAI language development and use are reviewed. Following this, the appropriateness of each of the three languages for authoing purposes is examined, and the main general design features and functions of a CAI authoring language are outlined. Economic and other considerations involved in the selection and implementation of a lexicon of operations are scrutinized. The conclusion is reached that it would seem fruitful to design a meta-language based on APL which would embody functions for the necessary instructional strategies and response analyses. It is stated tha such a language would greatly facilitate the use of CAI by the non-sophisticated teacher-author by allowing him to confine his activities to stating the content of his lesson and the strategy of his presentation. (PB)

REFLECTIONS ON CAI LANGUAGE DESIGN

Peter H. Braun

IBM Canada Laboratory

INTRODUCTION

There appears to be a need for a high level Computer Assisted Instruction (CAI) course authoring language. Grubb (1), for example, stated that CAI lacks a comprehensive notational system for describing and comparing instructional programs and providing new designs. This lack of a comprehensive notational system has sparked a variety of attempts at improving current authoring languages, and defining new high level authoring systems. For example,

> At the conference of the Association for the Development of Instructional Systems (ADIS), August 8-11, 1972, eleven of the twenty papers presented dealt with improved authoring methods, or proposals for advanced authoring systems, thus pointing to a pre-occupation with improving authoring methods among CAI users.
>
> Several agencies have recently advanced proposals for a universal high level authoring syntax. The most comprehensive efforts are those of the National Research Council, Canada, (September 1972), and the University of Freiburg, Germany, in cooperation with Simon Fraser University (since Fall, 1971).
>
> Also, it would appear that authoring systems based on Iverson's APL language are becoming increasingly popular. Noteworthy among these are the recent efforts at Cornell University (in cooperation with the IBM Systems Research Institute) on a system called ATS (A Terminal System), the University of Toronto (Erindale Campus) on a system called APL/CAT (Computer Assisted Tutorials), and (3) the CAL - system at Golden West College in Huntington Beach, California.
>
> The success of authoring systems based on APL rests primarily on the fact that the APL language is sufficiently flexible to permit a variety of unusually sophisticated approaches to answer analysis and response checking, thus permitting much more flexibility in student - computer dialog.
>
> In the "Index to Computer Assisted Instruction", (3) lists a total of 64 currently used different authoring languages and/or systems.
>
> To be sure, all but a dozen of these have only been used in local experimental situations. Yet, the mere fact that this many known

---

DR. PETER BRAUN received his Ph.D. in Educational Psychology from the University of Alberta, Edmonton. He is currently working on problems related to computer usage in education at the IBM Canada Laboratories in Toronto, Canada.

attempts at defining authoring languages have been made indicates that a universally acceptable authoring system does not yet seem to exist.

The above examples all emphasize the existence of a need for a high level authoring language.

The most commonly advanced rationale to support this contention is that CAI has lagged behind because the currently available languages require too much programming effort per hour of terminal instruction. Commonly used figures for writing CAI lessons based on Course-Writer range from ratios of 100 : 1 with simple materials, to 500 : 1 in the case of complex material. Not many teachers are willing to use a system which, once they spent time to master it, may still require upward of one hundred hours of preparation time for each hour of course material. Such a system becomes economically feasible only if the lessons so prepared will be useable by many students. Hence, CAI lessons must ideally be prepared such that they can easily and repeatedly be used by large numbers of students who might conceivably use a variety of different computers. In other words, economic feasibility of CAI depends on ease of exchangeability of programmed course material.

Since the future expansion of CAI depends primarily on the availability and exchangeability of course material, a high level CAI language which would ease the authoring burden per hour of course material, and permit the exchange of course material among CAI users would certainly increase CAI usage.

SOME HISTORICAL TRENDS OF CAI LANGUAGES

In order to establish future trends, it is often useful to look at what has happened in the past. For this reason, a breakdown of the 1264 CAI programs published by Lekan (3) was performed on the basis of the authoring language used. Table 1 lists all those languages having more than ten published programs in 1971. As can be seen, CWR, APL, and BASIC account for 66% of total usage, with CWR in the lead. This is, of course, not too surprising since CWR was specifically designed for the purpose of course authoring. However, the strength of both APL and BASIC is surprising because neither of them was designed for authoring purposes, and neither language has the built-in facilities for recording student responses which are a major feature of CWR. Hence, the reasons for their popularity lie elsewhere. Some possible reaons might be:

APL and (to a lesser degree) BASIC are easily learned in self-instructional fashion. The user can begin writing operational programs with minimal subsets of the full language. In contrast, CWR espouses a philosophy of organization which is not self-explanatory, and is difficult to understand by merely trying it on a terminal.

A second reason for the popularity of APL and BASIC is likely the mere availability of large numbers of terminals featuring one, or both of these languages. This sheer availability probably encourages the use of these languages for CAI purposes. Furthermore, the availability of APL and/or BASIC will increase much faster than that of CWR, because both languages are general purpose languages. This will likely further increase their share of CAI applications in the

future, and contribute to their phenomenal growth rate. Note, that in the 1970 Index to CAI, only 17 APL courses were published. The total number of published APL courses in 1971 was 228. Thus, almost the entire growth within CAI between 1970 and 1971 may be attributed to programs written in APL or BASIC. It appears that most of the effort in CWR was expended in converting existing CW I programs to CW II and/or CW III.

In summary, ease of usage, and availability of interactive terminals are likely the principal factors for the popularity of APL and BASIC as CAI authoring languages. The trend of using these languages for writing CAI courses will, therefore, accelerate.

As can be seen from Table I, FORTRAN and ALGOL are two general purpose languages which have at times been used for CAI course authoring while the remaining ones on the list are all specifically designed for authoring purposes only. However, none of them has become anywhere nearly as popular as the first three. This is not to say that they have no merits as CAI languages, and perhaps in due time one or the other of them may rise on the basis of special merit. Yet, without the explicit support of a major hardware manufacturer, these languages do not appear to hold much promise for advancing CAI, and hence they will not be discussed further.

COMPARISON OF CWR, APL, AND BASIC FOR AUTHORING PURPOSES

The three languages - CWR, APL, BASIC - are relatively similar in structure. Each provides a set of basic operations to the user who combines these to form executable programs. For authoring purposes, each of these languages is at a relatively low level and requires the composition of detailed instructions for the purpose of programming a CAI course. CWR has an initial advantage over APL and BASIC in that it performs many answer analysis operations containing implicit branches. Suitable functions would be required in APL and BASIC to emulate this advantage. This is, of course, no difficulty in either APL or BASIC, as both languages have the facility for generating user-specified functions. Indeed, such functions have been programmed in various locations. This point, however, illustrates one of the basic weaknesses of CWR: it is relatively difficult, and requires an experienced systems programmer, to add additional functions to CWR. Hence, most users quickly experience CWR as a closed system. It is not easily expansive, and, therefore, lacks a basic feature of good language design.

In addition, the number of functions implemented in CWR (primitive or systems defined functions) is much smaller than those available in BASIC or APL. In this respect, the APL language probably contains the most comprehensive set of primitive functions of any currently available programming language. This feature makes APL appear to the user as an open, or easily expansive system which is adaptable to the most demanding special purpose applications. BASIC occupies a middle ground between APL and CWR. Its main advantage is that it runs on very small systems (DEC, Hewlett Packard) having less than 22K of memory.

The number of functions provided by a given language is obviously related to its power of handling diverse problem situations. In terms of

TABLE I

LIST OF THE MOST FREQUENTLY USED LANGUAGES FOR CAI AUTHORING PURPOSES (3)

Language	# of Published CAI Courses* 1971	1970
CWR I/II/III	401	394
CW I	(48)	(147)
CW II	(260)	(213)
CW III	(93)	(34)
APL	228	17
BASIC	218	84
FORTRAN	96	43
TUTOR	73	118
CAILAN	31	31
PICLS	25	25
META SYMBOL	19	0
CAN	15	0
ALGOL	14	9
PLATO	13	58

*Lekan's (1971) CAI Index lists 1264 courses. The 1970 index lists 910.

language design, the functions provided by a language form a finite list of symbols (lexicon) from which patterned structures can be generated. The symbols are essentially analogus to the verbs of a natural language, and the patterned structures to the sentences formed from combining nouns and adjectives (the data) with the verbs (the operations) under the rules of grammar.

Note that the rules of grammar do not permit a random combination of the words. In order to be meaningful, the sentences formed by combining elements from the lexicon must adhere to certain grammatical patterns. However, if the lexicon is large, a large number of meaningful and syntactically correct structures can be generated. On the other hand, a small lexicon obviously limits the generation of many menaingful patterns.

GENERAL DESIGN FEATURES OF A CAI AUTHORING LANGUAGE

One can deal with the topic of design on at least two levels: a general level dealing with desirable design features, and a specific level dealing with the actual implementation. On the general level, one may produce a list of desirable features which, when scrutinized for details of implementation, may turn out to be economically unfeasible or impractical. A suitable compromise will then be required. However, the general features must be specified first in order to obtain a gross fit - an artist' sketch, so to speak - of the means to attain certain goals. These goals are elucidated next.

The primary purpose, or goal of any CAI language is to enable teachers to record their lessons in some systematic fasion which lends itself to translation into a machine readable and executable format. This is not as easy as it may sound because it requires precise definitions of the classroom environment and the teaching-learning process, and these are not readily available. The very nature of the classroom environment is often characterized by a certain degree of secretiveness, as much of it is conducted behind closed doors. Also, there is no uniform agreement about the most effective type of lesson presentation. A lesson usually consists of two distinct elements: the material to be taught, and the instructional strategy used by the teacher. Whereas the material may be relatively unchanging from occasion to occasion, or among instructors teaching the same course, the teaching strategies vary greatly. Often, they depend to some extent on the teacher's abilities and interests, the number and type of students in the class, as well as prevailing environmental conditions. Hence, it is this variability in teaching strategies which a CAI language must be able to capture if it is to be an effective agent of communication between teachers and students - using a computer as an interface.

While instructional strategies exist in many variations due to the aforementioned influence of teacher and student personality variables, there appears to be nevertheless a relatively small number of basic types of strategies in current use. For example, Grubb (1) analyzed a sample of 50 CAI programs on the basis of the instructional strategies used in these programs. (Unfortunately, his sample was not random, contained only 37 independent authors, and was limited exclusively to programs written in CWR II intended for use with the IBM 1500 system.) His analysis revealed that each of the 50 programs could be classified into a scheme of five categories on the basis of the prevalent instructional strategy used in each. These categories were roughly named:

Drill modes of interaction,
Practice,
Diagnosis and prescription,
Gaming and simulation,
Exploration.

These five patterns of instructional strategies, then, were derived empirically from the work of 37 teachers who have made the effort of recording their lessons in a medium suitable for execution on a computer, and, therefore, permitting public scrutiny of their work.

It may not be fair to say that these five patterns are the only ones possible. Without question, the constraints imposed by the IBM 1500 hardware and the CWR II language would limit the number of instructional strategies reproducible through these media. Yet, by using informed intuition rather than empirical data, Zinn (5) did not greatly surpass this list by enumerating the following seven types of basic strategies:

Drill
Teacher-controlled tutorial
Dialog tutorial
Simulation and gaming
Retrieval, and reorganization of information
Problem solving with computation
Artistic design and composition

As can be seen, the overlap between Zinn's and Grubb's lists is substantial. This then, would lend credence to the assertion that instructional strategies may be classifiable into a relatively small number of basic categories. However, due to combinations and permutations, a wide variety of "individualized" lessons is nevertheless possible.

A CAI-language, then, should minimally enable teachers to author programs using the known instructional strategies listed above, and, hopefully, be sufficiently flexible to allow possible expansion into presently undefined strategies. Zinn's list of categories covers the full spectrum of currently known data processing techniques, and perhaps some which have not yet been attempted. Obviously, a rather comprehensive lexicon of basic or primitive operations would be needed to satisfy such a variety of requirements.

Such a range of basic operations would, of course, be no more advantageous to use than currently available languages because it would require the detailed composition of programs by using merely a greater number of primitive operators. However, by using such a large set of basic operations, one could succeed to write programs which emulate all of the known instructional strategies. Once this has been done, one would only need to pass on different sets of data to these programs to generate various executable lesson modules. Since the number of known instructional strategies does not seem to be very large, the task of writing a suitable emulator, or "logic" for each of these strategies is within the realm of the possible. A teacher wanting to use such a system of logics would not need to acquire detailed knowledge of computer programming. He would merely have to organize his material in a fashion acceptable to the logic he wishes to use. Such a system, then, would greatly facilitate the creation of CAI lessons, and, simultaneously, would permit those teachers who are inclined in this direction to generate additional logics using the set of operators which are available.

Whereas nothing in the above proposal is really new, it deviates in one respect from earlier proposals or currently available systems featuring such function capability, namely, in respect to the specification that the set of primitive operators must be such that all currently known instructional strategies can be emulated. It is realized, of course, that such a specification may be unrealistic, and perhaps not attainable. Therefore, it is offered here as a desirable goal to be attained by a CAI-language.

A teacher's activity in the classroom is not confined only to teaching. He must also evaluate the effectiveness of his instructions. For this purpose, he must elicit, analyze, and record suitable feedback from the students. Hence, the facility of analyzing and recording student responses, and to summarize them for statistical purposes is another goal in the design of a CAI language. Particularly answer analysis requires much sophistication. Current CAI practice usually permits only minor deviation of the student's response from a pre-stored "correct response". This constraint often makes CAI programs appear rigid and unnatural.

The problems encountered in answer-analysis can be classified into at least three levels of complexity (4):

1. The arithmetic level
2. The logical level
3. The semantic level.

At the arithmetic level, an exact response is required which will match a pre-stored answer. For example, consider the question:

3 + 4 =

A correct answer would be any of the following:

7; SEVEN; seven; VII; 7.0

In any case, the answer analysis is simple since the student's response must match one element from a usually small set of elements. Note that multiple choice answers belong into this level.

On the logical level, the student's response can be infinitely variable. The only restriction is that the elements of the response must conform to a known set of rules. An example would be the response to the following problem:

Write an equation with one unknown.

As can be seen, the response analysis must be suitable to check if the submitted equation belongs to the class of equations having one unknown.

The semantic level is probably the most difficult. Consider this question posed by the computer:

What is a triangle?

The number of possible correct answers is very large. Some might be:

A closed figure with 3 sides
A closed figure having 3 sides
An area enclosed by 3 sides
A polygon having 3 sides
A polygon formed by 3 sides

Replacing the digit '3' with 'three' in the preceding answers doubles the list. Replacing "sides" with "segments" doubles it again, etc.

The above set does, of course, not exhaust the possibilities. Additional examples using different words might be:

3 points connected by lines
A polygon having three corners, etc.

The efficient analysis of such semantic responses requires techniques known as "approximate matching". A variety of such techniques have been developed. They usually rely on matching keywords, keyletters, or skeletons of keywords by means of some string-matching technique. By estimating the "goodness of fit" of the student's response, partial answers can also be dealt with effectively. For example, suppose the student responded to the above question as follows:

A triangle is a polygon.

A suitable answer from the computer should be:

It certainly is a polygon. But what makes a triangle different from other polygons? (etc.)

Unfortunately, one can almost always "beat" this system of semantic analysis by responding with exotic answers. But before criticizing the method for being imperfect, we should not forget that human communication also breaks down when one person responds to another in an insincere and intentionally exotic fashion. An evaluation or defence of these techniques is, however, not intended here. Rather, the above examples were cited to illustrate the complexity of various levels of answer analysis. The lexicon of available operations of a CAI language, then, must contain the necessary parts for assembling functions which can deal with such complex problems.

Lastly, CAI language design must include record keeping specifications. This is, of course, most easily accomplished with a computer, and, therefore, is a feature of all CAI languages. Yet, in their raw form, these responses are seldom useful. Hence, suitable functions are required which will summarize the recorded information. Such functions must yield statistical information pertaining to at least the following topics:

Evaluation of the course and its content. This is usually called formative evaluation.

Evaluation of a student's performance in the course. This is called summative evaluation.

Comparative evaluation of one or more classes of students.

In summary, a CAI language should contain functions which (1) emulate currently known teaching strategies, (2) permit response analyses at various levels of sophistication, and (3) record and summarize student responses in generally accepted fashions as required for evaluative purposes. Needless to say that such a language should embody a desirable feature of all good design, namely, the principle of parsimony.

SOME ECONOMIC CONSIDERATIONS

When considering the design of a CAI language, one can do a number of things:

1. One can start from scratch by defining the lexicon of necessary basic operations, implement them on a computer, and use them to build the required functions for the instructional strategy models.

2. One can use an existing (and implemented) lexicon of operations which has been used for related work in the past and appears to be suitable for building the required functions.

Both of these alternatives have been attempted in various locations. The best known attempts of defining and implementing a new lexicon of operations specifically intended for CAI authoring purposes are CWR (IBM), TUTOR and PLATO (University of Illinois), and CAN (Ontario Institute for Studies in Education).

Currently used authoring systems based on existing languages are ITS and CAILAN - both based on CWR, ATS, CAL, and CAT - based on APL; CATALYST based on BASIC; and METASYMBOL - based on FORTRAN.

In addition, a variety of recent attempts have been aimed at enhancing CWR to facilitate its usage by teachers who are unfamiliar with programming concepts, and to reduce the time required for writing CAI lessons. Some of these are:

A coursewriter pre-processor called VAULT (Versatile Authoring Language for Teachers) developed jointly by IBM and the University of Alberta, featuring separation of content and logic,

A variety of assembler language functions for CWR III, written and documented at Simon Fraser University, and a calculating function written at the University of Texas.

An interface between CWR and PL/1 implemented at Western Washington State University, permitting access to the best features of both languages.

A CWR-Macro system devised at Florida State University, being a direct attempt at using the CWR macro feature for generating modules to be used as building-blocks in instructional logics, and

An IBM program product called ITS (Interactive Training System) featuring separation of course content and logic for drill and practice programs such as occur most frequently in industrial training situations.

No attempt at completeness was intended with the above lists. They are merely useful to illustrate that the two approaches mentioned earlier - namely, starting either from scratch or with a defined lexicon - have both been attempted using a variety of different routes. The trade-off between choosing one or the other lies in the amount of effort needed to modify an existing lexicon for the special requirements of CAI. Generally speaking, however, a start from scratch involves considerably more work than using an existing lexicon as a basis. The advantage might be increased efficiency because the operators and functions can be specifically designed to serve the needs of CAI. However, with current hardware, this advantage is no longer a serious one, because on a large computer the amount of CPU time consumed during one hour of terminal time seldom exceeds three seconds. Even on small systems (e.g., IBM 1500), a student rarely consumes more than 30 seconds of CPU time per hour of sign-on time in CAI mode. It would be a mute exercise to expend much effort toward reducing these times. Hence, the more viable choice would be to base the design of a high level authoring language on an existing lexicon of symbols.

Economic considerations, then, would dictate that the use of an existing lexicon be at least seriously investigated before one would dismiss it as an unsuitable alternative. Whereas this task may at first glance appear to involve the investigation of a bewildering array of programming languages with regard to their suitability, this is not really so.

Since CAI has been around for several years now, one can make use of the knowledge of hundreds of CAI authors by simply looking at their language preference. Table 1 indicates that the overwhelming majority of authors appears to favour one of three languages for writing CAI programs, namely, CWR, APL, and/or BASIC. In fact, 66% of all CAI programs published until 1971 were written in one of these three languages, while the remaining 34% were spread over 61 other languages. Whereas there is a possibility that a real gem of a language exists among the 61 "others", the chance of an oversight by so many authors groping for a "better" language appears slim. In addition, one must consider the availability factor: at least one of the above three languages is available on a variety of widely marketed computers, while the gem might not be. Hence, practical considerations would dictate a choice from among CWR, APL, or BASIC as a foundation for a higher level authoring language.

CONSIDERATIONS FOR CHOOSING A LEXICON

At the extremes, a lexicon of operations may be either machine oriented, or problem oriented. The functional restrictions of the hardware available during the early 1960's forced the language designers of those days to accommodate their designs to such hardware restrictions as small memory sizes and limited amount of auxiliary storage. Hence, most languages dating back to that era are characterized by compromises toward machine orientation, rather than problem orientation. Since the CWR language originated at that time, it contains several examples of machine-oriented compromises - the most serious ones being the lack of floating-point arithmetic, severe limitations on the number of variables, the lack of branching into (and returning from) other available courses (e.g., CMI), and lack of any but the simplest level of response analysis.

At the other end of the machine versus problems oriented continuum is APL. In its original form, APL was devised as a means to improve the

precision of communication among scientists. It is, therefore, totally problem oriented, containing symbols for expressing all known arithmetic and logical operations, as well as a host of manipulative operations such as matching, merging, ranking, expanding, reducing, shaping, catenating, rotating, inverting, randomizing, dealing, selecting, reversing, transposing, decoding, and many more. In addition, combinations of these basic operations produce numerous new possibilities. For example, summation is achieved by combining the basic operations of addition and reduction. Even though the symbols in the APL-lexicon are precisely defined, their implementation on computers became feasible only after the hardware advanced to the technological level of the late 1960's. The problem orientation of APL is evident in the fact that its current (November 1972) implementation level offered as a program product by IBM lacks certain features which are distinctively and exclusively features associated with the manipulation of computer hardware rather than with the problem at hand. For example, at the present time, APL cannot generate or access data files used in other data processing activities, and it lacks a facility to change the internal representation of bit-strings into a variety of formats as required for numeric or alphameric manipulation, interpretation by the compiler, or formating of output. However, since several APL users (York University, I.P. Sharp and Assoc.) have succeeded to include these facilities into their APL systems, it would appear likely that future versions of APL will include these operators which permit hardware manipulations.

Due to its problem-orientation, APL has been received with great enthusiasm by those who can live within the above mentioned constraints. Particularly, the scientific and academic community has found that the language is an excellent vehicle for writing problem solving algorithms, as one needs not concern oneself with intricate rules based on hardware dependencies, but can concentrate totally on the problem. The language, therefore, appears much more "natural: than, for example, CWR, FORTRAN, etc. These latter languages continually force one to "fit" or "tailor" the problem into the constraints or dimensions imposed by the hardware, as reflected in the language, thus diverting attention from the problem at hand.

By contrast, the hardware is totally transparent to the APL user. This advantage is, of course, purchased at a price. Since APL is a problem oriented language implemented in interpretive mode, one cannot conveniently generate object decks. In other words, the program is interpreted every time it is executed. Benchmarks obtained with commercial programs indicate that an APL program may execute up to four times slower than an object program derived from a well-accepted conventional language, depending on the amount of "looping". Yet, as was discussed earlier, this is not a serious consideration in CAI, since the CPU-time for executing CAI programs is usually minute anyway, and there is very little looping.

BASIC, having been specifically designed for mini-computers, can be classified between CWR and APL. Its machine-orientation is apparent, but it does have the restrictions of CWR with respect to floating-point arithmetic, or the limitations on the number of variables. A good example illustrating the very detailed appearance of a CAI program using the NEW BASIC/CATALYST system can be found in Dwyer. (2)

It is apparent, then, that none of the three languages above is totally ideal. Each one has certain advantages, as well as serious disadvantages. The problems encountered in each language have been overcome in some

localities, but these solutions are usually not universally available. For example, floating-point routines for CWR have been written at the University of Texas, and by this author. However, the inclusion of these routines may slow down course-execution on small computers such as the IBM 1500. The remedies concerning APL have already been mentioned. Note that the resulting non-compatability of courses requiring the above mentioned "customizing" of the CAI-language produces an undesirable fragmentation of CAI efforts.

To arrive at a decision regarding the choice of a lexicon, one must then adopt a broader perspective than the mere weighing of current advantages versus disadvantages. Hence, an attempt is made here to look at the problem dynamically in a time frame, and to project some future trends.

Consider the fact that all the hardware constraints which produced the limitations in CWR no longer apply to current hardware, e.g., small memories and limited auxiliary storage. Consider further, that these same constraints formerly inhibited the introduction of APL. However, now that APL is feasible, its usage is increasing exponentially (user growth is 40% annually). It has been introduced into American schools ranging from public schools to universities. There are few universities where APL is not available, and several require facility with APL as part of their degree requirements, for example, all graduate students at UCLA's Graduate School of Management must show proficiency in APL as part of the requirements for the master's degree. APL is so easy to learn and to use that eighth grade students in Atlanta have been taught to write CAI modules in German, and to produce programs for simulating games. The March 1972 issue of the Computing Newsletter (University of Colorado) recently described APL as "the language of the 4th generation computers".

The prediction for APL, then, is certainly one of accelerating availability and usage. Also, as development costs increase and the cost of CPU time continues to decrease, the "pay-off" for switching to APL becomes more and more attractive for commercial users. This process will, of course, be gradual, and will not likely lead to the immediate obsolescence of other languages. However, if APL continues its present course of expansion, it will likely capture almost all new development effort by the end of the decade. This will be especially so in CAI, because the current user investment in established CAI languages is not of such magnitude as, for example, the investment in FORTRAN or COBOL. As Table 1 indicates, new CAI programs are already written in much greater quantity in APL than in CWR. Hence, one could expect that CWR usage may reach its peak in about 1975, and decline gradually from then on.

By contrast, BASIC requires a different set of considerations, because it addresses the mini-computer market. Neither CWR nor APL are available on mini-computers. Due to their low cost, sales of mini-computers are increasing. Consequently, one can expect increasing usage of BASIC in the near term future. With the introduction of virtual memory, and the rapid advances in semi-conductor memory technology, however, a sharp increase in the round-the-clock availability of time-sharing services which will greatly diminish the need for "in-house" mini-computers can be expected. This, then, will likely lead to a gradual decline in sales of mini-computers in the second half of the decade, and a comensurate reduction of the use of BASIC.

In summary, it has been shown that none of the three languages (CWR, APL, BASIC) is ideally suited for a CAI lexicon in their current state of implementation. CWR lacks floating-point arithmetic and in-core variable storage. APL lacks file access, and input-output formating features. BASIC is oriented toward record-processing, and would require special functions to handle many CAI applications. All of these shortcomings have been overcome through customized modifications and additions at one place or another. Hence, they are not insurmountable, making it difficult to base one's choice of a lexicon for a high level language on the technical merits alone.

Therefore, the problem was additionally viewed in a dynamic time-frame covering the decade of 1970-1980. In this context, it would appear that a number of circumstances will aid APL to continue its exponential user growth. These are primarily: ease of usage, adaptability of the language for writing problem solving algorithms, and increased "round-the-clock" availability of computers due to rapid developments in time-sharing techniques. Hence, APL will likely be the single most available lexicon for CAI purposes by the end of this decade. In view of this, it would seem most appropriate to use APL as a basis for a high level CAI language.

SUMMARY AND CONCLUSIONS

The preceding analysis showed that increasing numbers of CAI authors are choosing APL for writing CAI programs. In addition, several APL users have shown that a CAI meta-language based on APL is feasible (c.f., ATS - Cornell University; CAT - Erindale College; CAL - Golden West College). Hence, it would seem fruitful to design a meta-language based on APL which would embody functions for the various instructional strategies and differing types of response analyses discussed in this paper.

These functions should ideally be designed to accept appropriately arranged lesson material, as well as a set of suitable parameters pertaining to this material and the instructional model being used. Such a meta-language would greatly facilitate the use of CAI by a non-sophisticated teacher-author because most of the parameters could be defaulted until some expertise with entering instructional material into the system is gained. The aim is to make the computer transparent to the user, and to permit him ultimately to merely state the content of his lesson, and the strategy of presentation.

REFERENCES

1. Grubb, R. E. A design language and system for computer-assisted instruction programs. New York: IBM Education Research, and Columbia University, 1972.

2. Dwyer, T. A. Teacher/student authored CAI using the NEWBASIC system. Communications of the ACM, 1972, 15, 1, pp. 21-28.

3. Lekan, H. A. (Ed) Index to computer-assisted instruction (3 ed). New York: Harcourt, Brace, Jovanovich, 1971.

4. Peuchot, M. The problem of computer/student dialog. Pedagogic, 1968. 20, 5-6, pp. 504-515.

5. Zinn, K. L. Computer technology for teaching and research on instruction. Review of Ed. Research, 1967, 27, 5, pp. 618-634.